SPRAL_SSMFE
Sparse Symmetric Matrix-Free Eigensolver
Fortran User Guide

This package computes extreme (leftmost and/or rightmost) eigenpairs $\{\lambda_i, x_i\}$ of the following eigenvalue problems:

- the standard eigenvalue problem

$$Ax = \lambda x, \tag{1}$$

- the generalized eigenvalue problem

$$Ax = \lambda Bx, \tag{2}$$

- the buckling problem

$$Bx = \lambda Ax, \tag{3}$$

where A and B are **real symmetric** (or **Hermitian**) matrices and B is **positive definite**.

Evgueni Ovtchinnikov (STFC Rutherford Appleton Laboratory)

Major version history

2015-04-20 Version 1.0.0 Initial release

1 Installation

Please see the SPRAL install documentation. In particular note that:

- A BLAS library is required.
- A LAPACK library is required.

2 Usage overview

The eigensolver subroutines behind `SPRAL_SSMFE` implement a block iterative algorithm. The block nature of this algorithm allows the user to benefit from highly optimized linear algebra subroutines and from the ubiquitous multicore architecture of modern computers. It also makes this algorithm more reliable than Krylov-based algorithms employed e.g. by ARPACK in the presence of clustered eigenvalues. However, convergence of the iterations may be slow if the density of the spectrum is high.

Thus, good performance (in terms of speed) is contingent on the following two factors: (i) the number of desired eigenpairs must be substantial (e.g. not less than the number of CPU cores), and (ii) the employment of a convergence acceleration technique. The acceleration techniques that can be used are shift-and-invert and preconditioning. The former requires the direct solution of linear systems with the matrix A or its linear combination with B, for which a sparse symmetric indefinite solver (such as `HSL_MA97` or `SPRAL_SSIDS`) can

be employed. The latter applies to the case of positive definite A and requires a matrix or an operator[1] T, called *a preconditioner*, such that the vector $v = Tf$ is an approximation to the solution u of the system $Au = f$ (see a simple example in Section 7.1). This technique is more sophisticated and is likely to be of interest only to experienced users.

Additional options are offered by the packages SPRAL_SSMFE_EXPERT and SPRAL_SSMFE_CORE, upon which SPRAL_SSMFE is built and which are recommended for experienced users. Further information on the algorithm used by SPRAL_SSMFE can be found in the specification document for SPRAL_SSMFE_CORE and in Technical Report RAL-TR-2010-19.

2.1 Calling sequences

Access to the package requires a USE statement

```
use SPRAL_SSMFE
```

The following procedures are available to the user:

- ssmfe_standard() computes leftmost eigenpairs of (1), optionally using preconditioning
- ssmfe_standard_shift() computes eigenpairs of (1) near a given shift using the shift-and-invert technique
- ssmfe_generalized() computes leftmost eigenpairs of (2), optionally using preconditioning
- ssmfe_generalized_shift() computes eigenpairs of (2) near a given shift using the shift-and-invert technique
- ssmfe_buckling() computes eigenpairs of (3) near a given shift using the shift-and-invert technique
- ssmfe_free() should be called after all other calls are complete. It frees memory references by keep and inform.

The main solver procedures must be called repeatedly using a reverse communication interface. The procedure ssmfe_free() should be called once after the final call to a solver procedure to deallocate all arrays that have been allocated by the solver procedure.

2.2 Package types

INTEGER denotes default INTEGER, and REAL denotes REAL(kind=kind(0d0)). We use the term **call type** to mean REAL(kind=kind(0d0)) for calls to the double precision real valued interface, and to mean COMPLEX(kind=kind(0d0)) for calls to the double precision complex valued interface.

2.3 Derived types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types ssmfe_rcid (real version) or ssmfe_rciz (complex version), ssmfe_keepd (real version) or ssmfe_keepz (complex version), ssmfe_options and ssmfe_inform. The following pseudocode illustrates this.

```
use SPRAL_SSMFE
...
type (ssmfe_rcid   ) :: rcid
type (ssmfe_keepd  ) :: keepd
type (ssmfe_options) :: options
type (ssmfe_inform ) :: inform
...
```

The components of ssmfe_options and ssmfe_inform are explained in Sections 4.1 and 4.2. The components of ssmfe_keepd and ssmfe_keepz are used to pass private data between calls. The components of ssmfe_rcid and ssmfe_rciz that are used by SPRAL_SSMFE for the reverse communication are job, nx, ny, all of default INTEGER type, and x and y, which are two-dimensional arrays of call type.

[1] That is, an algorithm producing a vector $v = Tu$ for a given vector u.

3 Argument lists

3.1 ssmfe_standard(), ssmfe_standard_shift(), ssmfe_generalized(), ssmfe_generalized_shift(), and ssmfe_buckling()

To compute the leftmost eigenpairs of (1), optionally using preconditioning, the following call must be made repeatedly:

```
call ssmfe_standard( rci, left, mep, lambda, n, x, ldx, keep, options, inform )
```

To compute the eigenvalues of (1) in the vicinity of a given value sigma and the corresponding eigenvectors using the shift-and-invert technique, the following call must be made repeatedly:

```
call ssmfe_standard_shift &
     ( rci, sigma, left, right, mep, lambda, n, x, ldx, keep, options, inform )
```

To compute the leftmost eigenpairs of (2), optionally using preconditioning, the following call must be made repeatedly:

```
call ssmfe_generalized( rci, left, mep, lambda, n, x, ldx, keep, options, inform )
```

To compute the eigenvalues of (2) in the vicinity of a given value sigma and the corresponding eigenvectors using the shift-and-invert technique, the following call must be made repeatedly:

```
call ssmfe_generalized_shift &
     ( rci, sigma, left, right, mep, lambda, n, x, ldx, keep, options, inform )
```

To compute the eigenvalues of (3) in the vicinity of a given value sigma and the corresponding eigenvectors the following call must be made repeatedly:

```
call ssmfe_buckling &
     ( rci, sigma, left, right, n, mep, lambda, x, ldx, keep, options, inform )
```

`rci` is is an `INTENT(INOUT)` scalar of type `ssmfe_rcid` in the real version and `ssmfe_rciz` in the complex version. Before the first call, `rci%job` must be set to `0`. No other values may be assigned to `rci%job` by the user. After each call, the value of `rci%job` must be inspected by the user's code and the appropriate action taken (see below for details). The following values of `rci%job` are common to all solver procedures and require the same action:

- `-3` : fatal error return, the computation must be terminated;
- `-2` : non-fatal error return, the computation may be restarted, see Section 5 for the guidance;
- `-1` : the computation is complete and successful.
- `1` : the user must multiply the `n`×`rci%nx` matrix `rci%x(:)` by A and place the result into `rci%y(:)`.
- `2` : (`ssmfe_standard()` and `ssmfe_generalized()` only) the user must apply the preconditioner T to the `n`×`rci%nx` matrix `rci%x(:)` and place the result into `rci%y(:)`.
- `3` : (`ssmfe_generalized()`, `ssmfe_generalized_shift()` and `ssmfe_buckling()` only) the user must multiply the `n`×`rci%nx` matrix `rci%x(:)` by B and place the result into `rci%y(:)`.
- `9` : (`ssmfe_standard_shift()`, `ssmfe_generalized_shift()` and `ssmfe_buckling()` only) the solution of the shifted system with the right-hand side `n`×`rci%nx` matrix `rci%x(:)` must be placed into `rci%y(:)`. For problem (1), the shifted matrix is $A - \sigma I$, where I is $n \times n$ identity. For problem (2), the shifted matrix is $A - \sigma B$. For problem (3), the shifted matrix is $B - \sigma A$.

Restriction: `rci%job = 0` is the only value that can be assigned by the user.

`sigma` (`ssmfe_standard_shift()`, `ssmfe_generalized_shift()` and `ssmfe_buckling()` only) is an `INTENT(IN)` scalar of type `REAL` that holds the shift, a value around which the desired eigenvalues are situated.

`left` is an `INTENT(IN)` scalar of type default `INTEGER` that holds the number of desired leftmost eigenpairs. **Restriction:** $0 <$ `left + right` $\leq$ `min(mep, n/2)`, where `right` is zero for `ssmfe_standard()` and `ssmfe_generalized()`.

right (ssmfe_standard_shift(), ssmfe_generalized_shift() and ssmfe_buckling() only) is an INTENT(IN) scalar of type default INTEGER that holds the number of desired eigenvalues to the right of sigma. **Restriction:** $0 <$ left + right $\leq$ min(mep, n/2).

mep is an INTENT(IN) scalar of type default INTEGER that holds the size of the array lambda. See Section 6 for guidance on setting mep. **Restriction:** mep is not less than the number of desired eigenpairs (cf. left and right.

lambda(:) is an INTENT(INOUT) array of type REAL and size mep that is used to store the computed eigenvalues. After a successful completion of the computation it contains eigenvalues in ascending order. This array must not be changed by the user.

n is an INTENT(IN) scalar of type default INTEGER that holds the problem size. **Restriction:** n ≥ 1.

x(:,:) is an INTENT(INOUT) array of call type, and dimensions ldX and mep that is used to store the computed eigenvectors. The order of eigenvectors in x(:,:) is the same as the order of eigenvalues in lambda(:). This array may only be changed by the user before the first call to an eigensolver procedure (see the description of options%user_x in Section 4.1).

ldx is an INTENT(IN) scalar of type default INTEGER that holds the leading dimension of x(:,:). **Restriction:** ldx $\geq$ n.

keep is an INTENT(INOUT) scalar of type ssmfe_keepd in the real version and ssmfe_keepz in the complex version that holds private data.

options is an INTENT(IN) scalar of type ssmfe_options. Its components offer the user a range of options, see Section 4.1.

inform is an INTENT(INOUT) scalar of type ssmfe_inform. Its components provide information about the execution of the subroutine, see Section 4.2. It must not be changed by the user.

3.2 Terminating procedure

At the end of the computation, the memory allocated by the solver procedures should be released by making the following subroutine call:

```
ssmfe_free( keep, inform )
```

keep is an INTENT(INOUT) scalar of type ssmfe_keep, optional. On exit, its components that are allocatable arrays will have been deallocated.

inform is an INTENT(INOUT) scalar of type ssmfe_inform, optional. On exit, its components that are allocatable arrays will have been deallocated.

4 Derived types

4.1 type(ssmfe_options)

The derived data type ssmfe_options has the following components.

Convergence control options

abs_tol_lambda is a scalar of type REAL that holds an absolute tolerance used when testing the estimated eigenvalue error, see Section 6. The default value is 0. Negative values are treated as the default.

abs_tol_residual is a scalar of type REAL that holds an absolute tolerance used when testing the residual, see Section 6. The default value is 0. Negative values are treated as the default.

max_iterations is a scalar of type default INTEGER that contains the maximum number of iterations to be performed. The default value is 100.

`rel_tol_lambda` is a scalar of type `REAL` that holds a relative tolerance used when testing the estimated eigenvalue error, see Section 6. The default value is 0. Negative values are treated as the default.

`rel_tol_residual` is a scalar of type `REAL` that holds a relative tolerance used when testing the residual, see Section 6. If both `abs_tol_residual` and `rel_tol_residual` are set to 0, then the residual norms are not taken into consideration by the convergence test, see Section 6. The default value is 0. Negative values are treated as the default.

`tol_x` is a scalar of type `REAL` that holds a tolerance used when testing the estimated eigenvector error, see Section 6. If `tol_x` is set to zero, the eigenvector error is not estimated. If a negative value is assigned, the tolerance is set to `sqrt(epsilon(lambda))`. The default value is -1.0.

Printing options

`print_level` is a scalar of type default `INTEGER` that determines the amount of printing. Possible values are:

- < 0 : no printing;
- 0 : error and warning messages only;
- 1 : the type (standard or generalized) and the size of the problem, the number of eigenpairs requested, the error tolerances and the size of the subspace are printed before the iterations start;
- 2 : as 1 but, for each eigenpair tested for convergence (see Section 6), the iteration number, the index of the eigenpair, the eigenvalue, whether it has converged, the residual norm, and the error estimates are printed;
- > 2 : as 1 but with all eigenvalues, whether converged, residual norms and eigenvalue/eigenvector error estimates printed on each iteration.

The default value is 0. Note that for eigenpairs that are far from convergence, 'rough' error estimates are printed (the estimates that are actually used by the stopping criteria, see Section 6, only become available on the last few iterations).

`unit_error` is a scalar of type default `INTEGER` that holds the unit number for error messages. Printing is suppressed if `unit_error < 0`. The default value is 6.

`unit_diagnostic` is a scalar of type default `INTEGER` that holds the unit number for messages monitoring the convergence. Printing is suppressed if `unit_diagnostics < 0`. The default value is 6.

`unit_warning` is a scalar of type default `INTEGER` that holds the unit number for warning messages. Printing is suppressed if `unit_warning < 0`. The default value is 6.

Advanced options

`left_gap` is a scalar of type `REAL` that is only used when `left` is non-zero, and specifies the minimal acceptable distance between the last computed left eigenvalue and the rest of the spectrum. For `ssmfe_standard()` and `ssmfe_generalized()`, the last computed left eigenvalue is the rightmost of the computed ones, and for the other procedures it is the leftmost. If set to a negative value δ, the minimal distance is taken as $|\delta|$ times the average distance between the computed eigenvalues. Note that for this option to have any effect, the value of `mep` must be larger than `left + right`: see Section 6 for further explanation. The default value is 0.

`max_left` is a scalar of type default `INTEGER` that holds the number of eigenvalues to the left from σ, or a negative value, if this number is not known (cf. §6). The default is `max_left = -1`.

`max_right` is a scalar of type default `INTEGER` that holds the number of eigenvalues to the right from σ, or a negative value, if this number is not known. (cf. §6). The default is `max_right = -1`.

`right_gap` is a scalar of type `REAL` that is only used by `ssmfe_standard_shift()`, `ssmfe_generalized_shift()` and `ssmfe_buckling()` with a non-zero `right`, and has the same meaning as `options%left_gap` but for the rightmost computed eigenvalue. The default value is 0.

user_x is a scalar of type default INTEGER. If user_x > 0 then the first user_x columns of x(:,:) will be used as initial guesses for eigenvectors. Hence, if the user has good approximations to some of the required eigenvectors, the computation time may be reduced by putting these approximations into the first user_x columns of x(:,:). The default value is 0, i.e. the columns of x(:,:) are overwritten by the solver. **Restriction:** if user_x > 0 then the first user_x columns in x(:,:) must be linearly independent.

4.2 type(ssmfe_inform)

The derived data type ssmfe_inform is used to hold information from the execution of the solver procedures. The components are:

flag is a scalar of type default INTEGER that is used as an error flag. If a call is successful, flag has value 0. A nonzero value of flag indicates an error or a warning (see Section 5).

iteration is a scalar of type default INTEGER that holds the number of iterations.

left is a scalar of type default INTEGER that holds the number of converged eigenvalues on the left, i.e. the total number of converged eigenpairs for ssmfe_standard() and ssmfe_generalized(), and the number of the converged eigenvalues left of sigma for ssmfe_standard_shift(), ssmfe_generalized_shift() and ssmfe_buckling().

next_left is a scalar of type REAL that holds the non-converged eigenvalue next to the last converged on the left (cf. options%left_gap).

next_right is a scalar of type REAL that is used by ssmfe_standard_shift(), ssmfe_generalized_shift() and ssmfe_buckling() only, and holds the non-converged eigenvalue next to the last converged on the right (cf. options%right_gap).

non_converged is a scalar of type default INTEGER that holds the number of non-converged eigenpairs (see Section 5).

right is a scalar of type default INTEGER that is used by ssmfe_standard_shift(), ssmfe_generalized_shift() and ssmfe_buckling() only, and holds the number of converged eigenvalues right of sigma.

stat is a scalar of type default INTEGER that holds the allocation status (see Section 5).

5 Error flags

A successful return from a solver procedure is indicated by inform%flag = 0. A negative value indicates an error, a positive value indicates a warning; inform%data provides further information about some errors and warnings.
Possible negative values of inform%flag are as follows:

-1 rci%job is out-of-range.

-9 n is out-of-range.

-10 ldx is out-of-range.

-11 left is out-of-range.

-12 right is out-of-range.

-13 mep is less than the number of desired eigenpairs.

-100 Not enough memory; inform%stat contains the value of the Fortran stat parameter.

-200 B is not positive definite or user_x > 0 and linear dependent initial guesses were supplied.

Possible positive values are:

1 The iterations have been terminated because no further improvement in accuracy is possible (this may happen if B or the preconditioner is not positive definite, or if the components of the residual vectors are so small that the round-off errors make them essentially random). The value of `inform%non_converged` is set to the number of non-converged eigenpairs.

2 The maximum number of iterations `max_iterations` has been exceeded. The value of `inform%non_converged` is set to the number of non-converged eigenpairs.

3 The solver had run out of storage space for the converged eigenpairs before the gap in the spectrum required by `options%left_gap` and/or `options%right_gap` was reached. The value of `inform%non_converged` is set to the number of non-converged eigenpairs.

If the computation is terminated with the error code 2 or 3, it can be resumed with larger values of `max_iterations` and/or `mep`. In this case the user should set `options%user_X` to `info%left + info%right` and restart the reverse communication loop. An alternative option is to use one of the advanced solver procedures from `SPRAL_SSMFE_EXPERT` or `SPRAL_SSMFE_CORE` that delegate the storage of computed eigenpairs and the termination of the computation to the user.

6 Method

`SPRAL_SSMFE_CORE`, upon which `SPRAL_SSMFE` is built, implements a block iterative algorithm based on the Jacobi-conjugate preconditioned gradients (JCPG) method [2,3]. This algorithm simultaneously computes $m < n$ approximate eigenpairs, where the block size m exceeds the number n_e of desired eigenpairs for the sake of better convergence, namely, $m = n_e + \min(10, 0.1n_e)$.

An approximate eigenpair $\{x, \lambda\}$ is considered to have converged if the following three conditions are all satisfied:

1. if `options%abs_tol_lambda` and `options%rel_tol_lambda` are not both equal to zero, then the estimated error in the approximate eigenvalue must be less than or equal to

   `max(options%abs_tol_lambda,` δ`*options%rel_tol_lambda)`,

   where δ is the estimated average distance between eigenvalues.

2. if `options%tol_x` is not zero, then the estimated sine of the angle between the approximate eigenvector and the invariant subspace corresponding to the eigenvalue approximated by λ must be less than or equal to `options%tol_x`.

3. if `options%abs_tol_residual` and `options%rel_tol_residual` are not both equal to zero, then the Euclidean norm of the residual, $\|Ax - \lambda Bx\|_2$, must be less than or equal to

   `max(options%abs_tol_residual, options%rel_tol_residual*`$\|\lambda Bx\|_2$`)`.

The extra eigenpairs are not checked for convergence, as their role is purely auxiliary.

If the gap between the last computed eigenvalue and the rest of the spectrum is small, then the accuracy of the corresponding eigenvector may be very low. To prevent this from happening, the user should set the eigenpairs storage size `mep` to a value that is larger than the number of desired eigenpairs, and set the options `options%left_gap` and `options%right_gap` to non-zero values δ_l and δ_r. These values determine the size of the minimal acceptable gaps between the computed eigenvalues and the rest of the spectrum, δ_l referring to either leftmost eigenvalues (for `ssmfe_standard()` and `ssmfe_generalized()` only) or those to the left of the shift `sigma`, and δ_r to those to the right of the shift `sigma`. Positive values of δ_l and δ_r set the gap explicitely, and negative values require the gap to be not less than their absolute value times the average distance between the computed eigenvalues. A recommended value of δ_l and δ_r is -0.1. The value of `mep` has little effect on the speed of computation, hence it might be set to any reasonably large value. The larger the value of `mep`, the larger the size of an eigenvalue cluster for which accurate eigenvectors can be computed, notably: to safeguard against clusters of size up to k, it is sufficient to set `mep` to the number of desired eigenpairs plus $k - 1$.

When using the solver procedures that employ the shift-and-invert technique, it is very important to ensure that the numbers of desired eigenvalues each side of the shift do not exceed the actual numbers of these eigenvalues, as the eigenpairs 'approximating' non-existing eigenpairs of the problem will not converge. It is therefore strongly recommended that the user employs a linear system solver that performs the LDLT

factorization of the shifted system, e.g. HSL_MA97 or SPRAL_SSIDS. The LDLT factorization of the matrix $A - \sigma B$ consists in finding a lower triangular matrix L, a block-diagonal matrix D with 1×1 and 2×2 blocks on the main diagonal and a permutation matrix P such that $P^T(A - \sigma B)P = LDL^T$. By inertia theorem, the number of eigenvalues to the left and right from the shift σ is equal to the number of negative and positive eigenvalues of D, which allows quick computation of the eigenvalue numbers each side of the shift.

References

[1] E. E. Ovtchinnikov and J. Reid. A preconditioned block conjugate gradient algorithm for computing extreme eigenpairs of symmetric and Hermitian problems. Technical Report RAL-TR-2010-19, 2010.
[2] E. E. Ovtchinnikov, *Jacobi correction equation, line search and conjugate gradients in Hermitian eigenvalue computation I: Computing an extreme eigenvalue*, SIAM J. Numer. Anal., **46**:2567–2592, 2008.
[3] E. E. Ovtchinnikov, *Jacobi correction equation, line search and conjugate gradients in Hermitian eigenvalue computation II: Computing several extreme eigenvalues*, SIAM J. Numer. Anal., **46**:2593–2619, 2008.

7 Examples

7.1 Preconditioning example

The following code computes the 5 leftmost eigenpairs of the matrix A of order 100 that approximates the two-dimensional Laplacian operator on a 20-by-20 grid. One forward and one backward Gauss-Seidel update are used for preconditioning, which halves the number of iterations compared with solving the same problem without preconditioning. The module laplace2d (examples/Fortran/ssmfe/laplace2d.f90) supplies a subroutine apply_laplacian() that multiplies a block of vectors by A, and a subroutine apply_gauss_seidel_step() that computes $y = Tx$ for a block of vectors x by applying one forward and one backward update of the Gauss-Seidel method to the system $Ay = x$.

```
! examples/Fortran/ssmfe/precond_ssmfe.f90
! Laplacian on a square grid (using SPRAL_SSMFE routines)
program ssmfe_precond_example
  use spral_ssmfe
  use laplace2d ! implement Lapalacian and preconditioners
  implicit none

  integer, parameter :: wp = kind(0d0) ! Working precision is double

  integer, parameter :: m   = 20    ! grid points along each side
  integer, parameter :: n   = m*m   ! problem size
  integer, parameter :: nep = 5     ! eigenpairs wanted

  real(wp) :: lambda(2*nep)         ! eigenvalues
  real(wp) :: X(n, 2*nep)           ! eigenvectors
  type(ssmfe_rcid   ) :: rci        ! reverse communication data
  type(ssmfe_options) :: options    ! options
  type(ssmfe_keepd  ) :: keep       ! private data
  type(ssmfe_inform ) :: inform     ! information
  integer :: i                      ! loop index

  ! the gap between the last converged eigenvalue and the rest of the spectrum
  ! must be at least 0.1 times average gap between computed eigenvalues
  options%left_gap = -0.1
  rci%job = 0
  do ! reverse communication loop
    call ssmfe_standard &
      ( rci, nep, 2*nep, lambda, n, X, n, keep, options, inform )
    select case ( rci%job )
```

```
    case ( 1 )
      call apply_laplacian( m, m, rci%nx, rci%x, rci%y )
    case ( 2 )
      call apply_gauss_seidel_step( m, m, rci%nx, rci%x, rci%y )
    case ( :-1 )
      exit
    end select
  end do
  print '(i3, 1x, a, i3, 1x, a)', inform%left, 'eigenpairs converged in', &
     inform%iteration, 'iterations'
  print '(1x, a, i2, a, es13.7)', &
    ('lambda(', i, ') = ', lambda(i), i = 1, inform%left)
  call ssmfe_free( keep, inform )
end program ssmfe_precond_example
```

This code produces the following output:

```
  6 eigenpairs converged in 19 iterations
 lambda( 1) = 4.4676695E-02
 lambda( 2) = 1.1119274E-01
 lambda( 3) = 1.1119274E-01
 lambda( 4) = 1.7770878E-01
 lambda( 5) = 2.2040061E-01
 lambda( 6) = 2.2040061E-01
```

Note that the code computed one extra eigenpair because of the insufficient gap between the 5th and 6th eigenvalues.

7.2 Shift-and-invert example

The following code computes the eigenpairs of the matrix of order 64 that approximates the two-dimensional Laplacian operator on 8-by-8 grid with eigenvalues near the shift `sigma` = 1.0. For the shifted solve, LAPACK subroutines `DSYTRS` and `DSYTRF` are used, which perform the LDLT-factorization and the solution of the factorized system respectively. The matrix of the discretized Laplacian is computed by the subroutine `set_2d_laplacian_matrix()` from the `laplace2d` module (`examples/Fortran/ssmfe/laplace2d.f90`). The module `ldltf` (`examples/Fortran/ssmfe/ldltf.f90`) supplies the function `num_neg_D()` that counts the number of negative eigenvalues of the D-factor.

```
! examples/Fortran/ssmfe/shift_invert.f90
! Laplacian on a rectangular grid by shift-invert via LDLT factorization
program ssmfe_shift_invert_example
  use spral_ssmfe
  use laplace2d ! implement Lapalacian and preconditioners
  use ldltf     ! implements LDLT support routines
  implicit none

  integer, parameter :: wp = kind(0d0) ! Working precision

  integer, parameter :: nx = 8    ! grid points along x
  integer, parameter :: ny = 8    ! grid points along y
  integer, parameter :: n = nx*ny ! problem size
  real(wp), parameter :: sigma = 1.0 ! shift

  integer :: ipiv(n)               ! LDLT pivot index
  real(wp) :: lambda(n)            ! eigenvalues
  real(wp) :: X(n, n)              ! eigenvectors
  real(wp) :: A(n, n)              ! matrix
  real(wp) :: LDLT(n, n)           ! factors
```

```
  real(wp) :: work(n*n)            ! work array for dsytrf
  integer :: lwork = n*n           ! size of work
  integer :: left, right           ! wanted eigenvalues left and right of sigma
  integer :: i                     ! index
  type(ssmfe_options) :: options   ! eigensolver options
  type(ssmfe_inform ) :: inform    ! information
  type(ssmfe_rcid   ) :: rci       ! reverse communication data
  type(ssmfe_keepd  ) :: keep      ! private data

  call set_laplacian_matrix( nx, ny, A, n )

  ! perform LDLT factorization of the shifted matrix
  LDLT = A
  forall ( i = 1 : n ) LDLT(i, i) = A(i, i) - sigma
  lwork = n*n
  call dsytrf( 'L', n, LDLT, n, ipiv, work, lwork, i )

  left = num_neg_D(n, LDLT, n, ipiv) ! all eigenvalues to the left from sigma
  right = 5                          !   5 eigenvalues to the right from sigma
  rci%job = 0
  do
    call ssmfe_standard_shift &
      ( rci, sigma, left, right, n, lambda, n, X, n, keep, options, inform )
    select case ( rci%job )
    case ( 1 )
      call dgemm &
        ( 'N', 'N', n, rci%nx, n, 1.0_wp, A, n, rci%x, n, 0.0_wp, rci%y, n )
    case ( 9 )
      call dcopy( n * rci%nx, rci%x, 1, rci%y, 1 )
      call dsytrs( 'L', n, rci%nx, LDLT, n, ipiv, rci%y, n, i )
    case ( :-1 )
      exit
    end select
  end do
  print '(1x, a, es10.2, 1x, a, i3, 1x, a)', 'Eigenvalues near', sigma, &
     '(took', inform%iteration, 'iterations)'
  print '(1x, a, i2, a, es13.7)', &
    ('lambda(', i, ') = ', lambda(i), i = 1, inform%left + inform%right)
  call ssmfe_free( keep, inform )
end program ssmfe_shift_invert_example
```

This code produces the following output:

```
 Eigenvalues near  1.00E+00 (took  5 iterations)
 lambda( 1) = 2.4122952E-01
 lambda( 2) = 5.8852587E-01
 lambda( 3) = 5.8852587E-01
 lambda( 4) = 9.3582223E-01
 lambda( 5) = 1.1206148E+00
 lambda( 6) = 1.1206148E+00
 lambda( 7) = 1.4679111E+00
 lambda( 8) = 1.4679111E+00
 lambda( 9) = 1.7733184E+00
```

7.3 Hermitian example

The following code computes the 5 leftmost eigenpairs of the differential operator $i\frac{d}{dx}$ acting in the space of periodic functions discretized by central differences on a uniform mesh of 80 steps.

```
! examples/Fortran/ssmfe/hermitian.f90 - Example code for SPRAL_SSMFE package
! Hermitian operator example
program ssmfe_hermitian_example
  use spral_ssmfe
  implicit none

  integer, parameter :: wp = kind(0d0) ! working precision

  integer, parameter :: n   = 80     ! problem size
  integer, parameter :: nep = 5      ! eigenpairs wanted

  real(wp) :: lambda(nep)            ! eigenvalues
  complex(wp) :: X(n, nep)           ! eigenvectors
  type(ssmfe_rciz   ) :: rci         ! reverse communication data
  type(ssmfe_options) :: options     ! options
  type(ssmfe_keepz  ) :: keep        ! private data
  type(ssmfe_inform ) :: inform      ! information
  integer :: i                       ! loop index

  rci%job = 0
  do ! reverse communication loop
    call ssmfe_standard( rci, nep, nep, lambda, n, X, n, keep, options, inform )
    select case ( rci%job )
    case ( 1 )
      call apply_idx( n, rci%nx, rci%x, rci%y )
    case ( :-1 )
      exit
    end select
  end do
  print '(i3, 1x, a, i3, 1x, a)', inform%left, 'eigenpairs converged in', &
     inform%iteration, 'iterations'
  print '(1x, a, i2, a, es14.7)', &
    ('lambda(', i, ') = ', lambda(i), i = 1, inform%left)
  call ssmfe_free( keep, inform )

contains

  subroutine apply_idx( n, m, x, y ) ! central differences for i d/dx
    implicit none
    complex(wp), parameter :: IM_ONE = (0.0D0, 1.0D0)
    integer, intent(in) :: n, m
    complex(wp), intent(in) :: x(n, m)
    complex(wp), intent(out) :: y(n, m)
    integer :: i, j, il, ir
    do j = 1, m
      do i = 1, n
        if ( i == 1 ) then
          il = n
        else
          il = i - 1
        end if
        if ( i == n ) then
          ir = 1
        else
          ir = i + 1
        end if
```

```
        y(i, j) = IM_ONE*(x(ir, j) - x(il, j))
      end do
    end do
  end subroutine apply_idx

end program ssmfe_hermitian_example
```

This code produces the following output:

```
  5 eigenpairs converged in 25 iterations
 lambda( 1) = -2.0000000E+00
 lambda( 2) = -1.9938347E+00
 lambda( 3) = -1.9938347E+00
 lambda( 4) = -1.9753767E+00
 lambda( 5) = -1.9753767E+00
```